REMARKS

In an Office Action dated May 18, 2006, the Examiner rejected claims 13-19 under 35 U.S.C. §101 as directed to non-statutory subject matter; rejected claims 1-5, 7, 8 13-17 and 19 under 35 U.S.C. §102(b) as anticipated by Smith, et al. (US 6,311,324); rejected claims 6 and 18 under 35 U.S.C. §103(a) as unpatentable over *Smith* in view of Blume (US 6,223,337); rejected claims 9 and 11 under 35 U.S.C. §103(a) as unpatentable over *Smith* in view of Hunt (US 6,499,137); and rejected claims 10 and 12 under 35 U.S.C. §103(a) as unpatentable over *Smith* in view of *Hunt* and Bates, et al. (US 6,981,248).

Non-Statutory Subject Matter

Claim 13 has been amended to recite that the instructions are recorded on *tangible* signal bearing media, and claim 13, as well as those claims dependent on it, is therefore statutory.

Prior Art

Applicants have amended all independent claims to clarify the claimed invention. In particular, the claims have been amended to clarify that the selective determination and certain other steps are performed by an automated compiler (claims 1 and 8), and that the compiler foregoes certain possible optimizations when making the selective determination (claims 1 and 13). As amended, the claims are patentable over the cited art.

A brief background discussion is in order to appreciate applicants' invention. It is well known in the art for compilers to perform certain optimizations of computer programming source code in order to improve performance. I.e., optimizing compilers will sometimes reorder code statements, combine or split up operations, pre-compute certain constant coefficients in operations, and so forth, in order to achieve greater execution efficiency. When source code is compiled without optimization, there is generally a one-to-one correspondence between each line

of source code and some discrete set of lines of resultant object code. Therefore, a listing of compiled object code is directly mappable to the source code lines which produced it. This often simplifies the debug process, for if an abnormal condition occurs at any point during execution or a break point is encountered, it is possible to immediately identify the corresponding source code statement and location in the source code. However, when code is optimized by the compiler during compilation, the one-to-one mapping between object code segments and source code statements is lost. It is not always possible to identify the source code statement to which an object code statement corresponds, and even if identified, the programmer can not be assured that all statements in the vicinity have been executed in order as they appear in the source code.

The difficulty in debugging optimized object code has long been known, and for this reason, among others, most compilers provide the user with an option to optimize or not to optimize. When code is under development and debug activity is frequent, the programmer will typically turn off optimization so that it is easier to identify bugs in the code. Once the programmer is fairly confident that the code is stable, he will produce an optimized compiled version for shipment to the ultimate users.

However, this makes it extremely difficult to debug certain errors discovered in the field, after shipment to the ultimate user. Applicants observed that, since code often goes through many versions and iterations, often over a period of years, in any one program there may be very stable portions of the code which have been in use for a relatively long time, and other portions which are modified more frequently, or have only recently been added, and are therefore less stable. Applicants therefore propose to selectively optimize the less stable portions of the code. An automated compiler can use debug history data or other data to identify those portions which are less stable, and to optimize only the more stable portions. Thus, the resultant object code has the advantage of serviceability for those code portions which are likely to need it, while being optimized for execution performance for those code portions which have demonstrated stability.

The key feature of applicants' invention is therefore that the compiler automatically determines whether to optimize selective portions of the code, optimizing some and not others, even though it has the capability to optimize all (or none).

Smith discloses a tuning advisor for recommending that the programmer consider performing certain improvements to the source code. These are sometimes called "optimizations", but is should be clear that these are not optimizations performed by the compiler. They are source code improvements performed by the programmer. In some cases, a compiler or similar automated program can identify potential optimizations, but because it makes certain conservative assumptions, it can not perform these optimizations automatically. By alerting the programmer to such potential optimizations, the programmer is enabled to make them himself, i.e., by modifying the source code.

The Examiner's rejection, as nearly as applicants can understand it, appears to be based on the idea that the programmer makes a decision to perform certain optimizations, and these optimizations are ultimately compiled with a compiler. While this was not the intention of the claim language "with an automated compiler", applicants have nevertheless clarified claims 1 and 8 to recite that the compilation steps are performed by the automated compiler. In particular, the step of selectively determining whether to optimize a discrete component portion is performed by the automated compiler. Applicants have not made a corresponding amendment to claim 13, because, that claim being a program product claim which recites that the steps are performed by the program, the limitation that the recited steps are performed by an automated process is inherent in the claim.

Furthermore, it appears the Examiner may be reading the recited steps on the conventional compilation process, i.e., a compiler determines whether it is possible to optimize some series of statements, and if so, performs the optimization. Applicants have therefore amended claims 1 and

13 to clarify that the selective determination is with respect to optimizations the compiler could perform, i.e., it foregoes at least one optimization which it is capable of performing as a result of the selective determination.

Representative claim 1, as amended, recites:

1. A method for compiling computer programming code, comprising the steps of:
generating a compilable source module, said source module containing a plurality of discrete component portions;
generating selective optimization data, said selective optimization data including a plurality of selective optimization data portions, each of said plurality of selective optimization data portions corresponding to a respective component portion of said plurality of discrete component portions; and
compiling said compilable source module with an automated compiler, wherein said compiling step comprises *the following steps performed by said automated compiler*:
(a) with respect to each of said plurality of discrete component portions, *selectively determining whether to optimize the respective discrete component portion using said selective optimization data portion* corresponding to the respective discrete component portion;
(b) with respect to at least one discrete component portion of a first subset of said plurality of discrete component portions, said first subset containing the discrete component portions for which said selectively determining step determined to optimize the respective discrete component portion, *performing at least one optimization upon the respective discrete component portion responsive to said selectively determining step*; and
(c) with respect to at least one discrete component portion of a second subset of said plurality of discrete component portions, said second subset containing the discrete component portions for which said selectively determining step determined not to optimize the respective discrete component portion, *compiling the respective discrete component portion without performing at least one optimization which said automated compiler has the capability to automatically perform* on the respective discrete component portion. [emphasis added]

Claim 13 is a program product claim of analogous scope. Claim 8 is a method claim which varies in certain respects, and in particular recites the use of debug activity data for making the selective determination.

The selective determinations made in *Smith* are entirely manual in nature, and are not responsive to debug activity data. I.e., *Smith*'s tuning advisor flags possible optimizations, but the programmer must then alter the source code if the optimization is to be performed. Therefore the recited selective determination step is not met by *Smith*.

Furthermore, *Smith* does not teach or suggest the use of "debug activity data" for any purpose (see claim 8). The Examiner apparently considers the advice generated by the tuning advisor to be "debug activity data" Applicants disagree. The tuning advice does not record or relate to any debug ***activity***. It is used for performance improvement. In a very general sense, performance improvement is related to debug. But it is not a record of ***debug activity***. I.e., the tuning advice does not show any past actions taken with respect to ***debug activity***, such as code changes, execution of debug utilities, errors occurring in the code, and so forth.

For all the reasons stated above, the claims as amended are not anticipated by *Smith*.

For similar reasons, the claims are not obvious over *Smith*, either alone or in combination with other cited art. *Smith* is directed to an entirely different purpose, one of providing tuning advice to the programmer so that he may make certain performance improvements to the source code. *Smith* neither discloses the existence of the problem to which applicants' invention is directed, nor suggests a solution. In particular, there is no discussion whatsoever in *Smith* of the difficulty of servicing optimized code, and the serviceability differences between optimized and non-optimized code. There is no motivation shown in *Smith* for altering an optimizing compiler, so that it will automatically determine whether to optimize selective code portions using debug activity or other optimization data, and forego at least some optimizations that could otherwise have been performed.

The secondary references likewise fail to teach or suggest the essential elements of applicants' claimed invention. *Blume* discloses a compiler in which optimization can be turned off, as is well known in the art, but does not teach or suggest turning compilation off with respect to selective code portions. *Hunt* discloses a method for dynamic linking, and is cited to show the use of counters to record debug events, but similarly does not teach or suggest applicants' essential features. *Bates* discloses the use of conditional breakpoints for debug, and is cited to show the use of variable visualization counters to record debug events. Like *Hunt*, it too does not teach or suggest the essential features of applicants' invention.

In view of the foregoing, applicants submit that the claims are now in condition for allowance, and respectfully request reconsideration and allowance of all claims. In addition, the Examiner is encouraged to contact applicants' attorney by telephone if there are outstanding issues left to be resolved to place this case in condition for allowance.

Respectfully submitted,

JOHN M. ADOLPHSON, et al.

By:_____
Roy W. Truelson
Registration No. 34,265

Telephone: (507) 202-8725